

Luis Antonio Ortega Andrés 13/01/2021

```
In [1]: %load_ext autoreload
%autoreload 2
import numpy as np
import matplotlib.pyplot as plt
import stochastic_plots as stoch
import examen_ordinaria_PE_2020_2021 as pe
import scipy as sp
import BM_simulators as BM

# Author: <alberto.suarez@uam.es> <luisa.ortega@estudiante.uam.es>
```

## Exercise 1: Simulation of a Continuous-time Markov Chain

a) La matriz de transición del proceso de saltos subyacente es

$$\tilde{P} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0.5 & 0.5 & 0 \end{pmatrix}$$

b) Calculamos la distribución estacionaria  $(\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_3)$  sabiendo que verifica

$$(\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_3)\tilde{P} = (\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_3).$$

Esto nos deja las relaciones:

$$\begin{cases} \tilde{\pi}_1 = \frac{1}{2}\tilde{\pi}_3 \\ \tilde{\pi}_2 = \frac{1}{2}\tilde{\pi}_3 + \tilde{\pi}_1 \\ \tilde{\pi}_3 = \tilde{\pi}_2 \end{cases}$$

Si utilizamos además que  $\tilde{\pi}_1 + \tilde{\pi}_2 + \tilde{\pi}_3 = 1$ , tenemos que

$$\tilde{\pi}_1 + \tilde{\pi}_2 + \tilde{\pi}_3 = \frac{1}{2}\tilde{\pi}_3 + \tilde{\pi}_3 + \tilde{\pi}_3 = 1 \implies \tilde{\pi}_3 = \frac{2}{5}.$$

Luego  $\tilde{\pi}_1 = \frac{1}{5}$  y  $\tilde{\pi}_2 = \frac{2}{5}$ .

c) Utilizamos que la distribución estacionaria de la cadena de Markov en tiempo continuo  $(\pi_1, \pi_2, \pi_3)$  vienen caracterizados por

$$\pi_i \propto \frac{\tilde{\pi}_i}{\lambda_i}, i = 1, 2, 3 \text{ y } \pi_1 + \pi_2 + \pi_3 = 1.$$

Tenemos entonces,

$$\begin{cases} \pi_1 \propto \frac{\tilde{\pi}_1}{\lambda_1} = \frac{1/5}{2} = \frac{1}{10}, \\ \pi_2 \propto \frac{\tilde{\pi}_2}{\lambda_2} = \frac{2/5}{1} = \frac{2}{5}, \\ \pi_3 \propto \frac{\tilde{\pi}_3}{\lambda_3} = \frac{2/5}{3} = \frac{2}{15}, \end{cases}$$

Tras normalizar obtenemos:

$$\begin{cases} \pi_1 = \frac{3}{19}, \\ \pi_2 = \frac{12}{19}, \\ \pi_3 = \frac{4}{19}. \end{cases}$$

**d)** Utilizamos que cada elemento de la matriz generadora infinitesimal  $G$  cumple:

$$g_{i,j} = \begin{cases} \lambda_i p_{i,j}, & i \neq j \\ -\lambda_i & i = j \end{cases}.$$

Tenemos entonces

$$G = \begin{pmatrix} -2 & 2 & 0 \\ 0 & -1 & 1 \\ 1.5 & 1.5 & -3 \end{pmatrix}.$$

**e)** Utilizamos que la distribución estacionaria verifica

$$(\pi_1 \quad \pi_2 \quad \pi_3) G = 0.$$

Esto nos genera el siguiente sistema de ecuaciones:

$$\begin{cases} -2\pi_1 + 1.5\pi_3 = 0, \\ 2\pi_1 - \pi_2 + 1.5\pi_3 = 0, \\ \pi_2 - 3\pi_3 = 0. \end{cases} \implies \begin{cases} \pi_2 = 3\pi_3 \\ 2\pi_1 = \frac{3}{2}\pi_3 \end{cases}$$

Vemos que el vector que obtuvimos en el apartado **c)** cumple estas mismas ecuaciones, de forma que (debido a la normalidad) el resultado es el mismo que obtuvimos en dicho apartado.

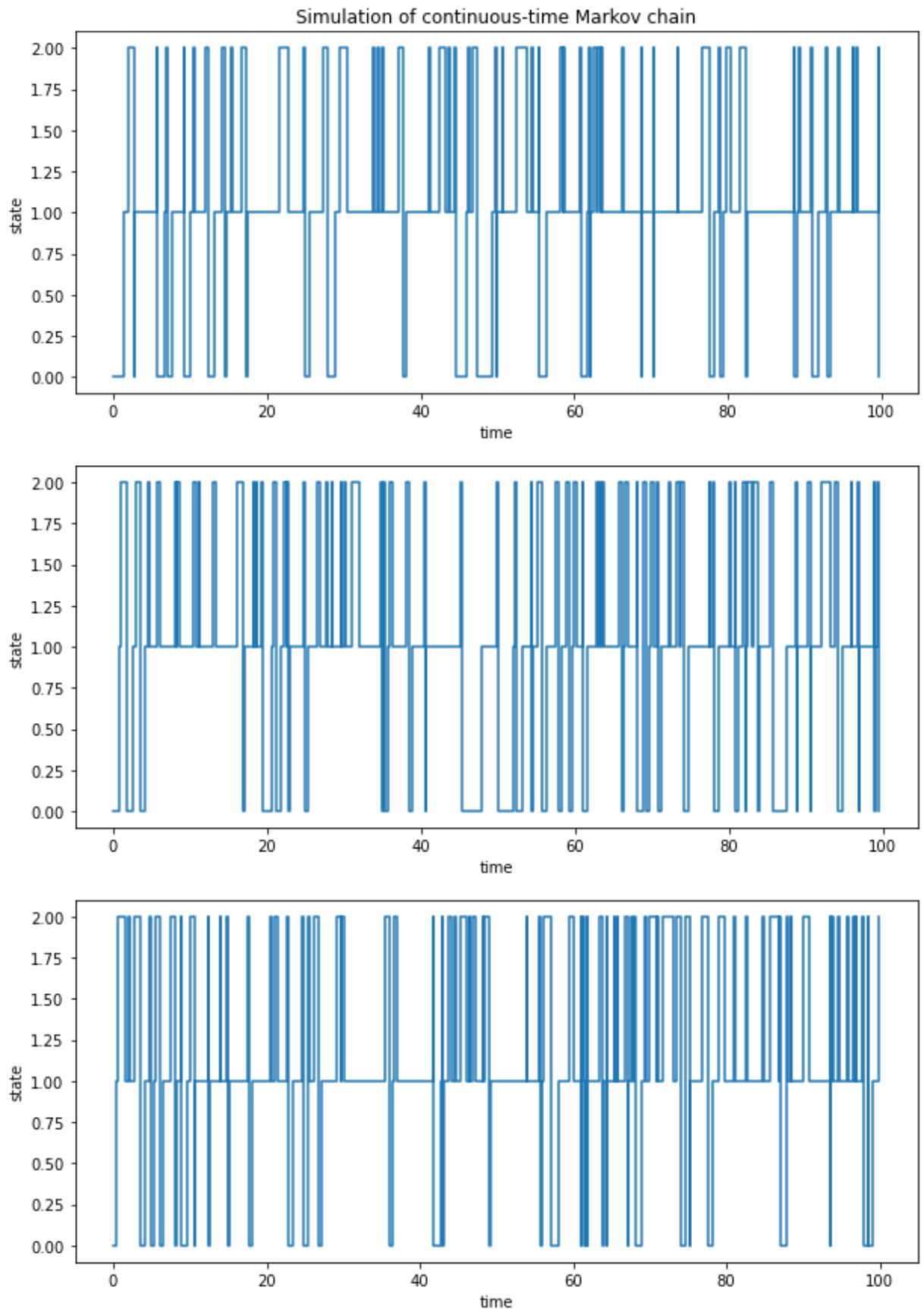
**f)** Simula trayectorias del proceso para suponiendo que en el instante  $t = 0$  el sistema se encuentra en  $S_0 = 0$ .

Simulamos  $M = 3$  trayectorias distintas.

```
In [2]: # Constant values
transition_matrix = [[0, 1, 0], [0, 0, 1], [1/2, 1/2, 0]]
lambda_rates = [2, 1, 3]
t0 = 0.0
t1 = 100.0
state_0 = 0
M = 3

arrival_times_CTMT, trajectories_CTMT = (
    pe.simulate_continuous_time_Markov_Chain(
        transition_matrix, lambda_rates,
        state_0, M, t0, t1))

# Simulation of a few trajectories
fig, ax = plt.subplots(3,1, figsize=(10, 15), num = 1)
ax[0].set_title("Simulation of continuous-time Markov chain")
for i in range(M):
    ax[i].step(arrival_times_CTMT[i], trajectories_CTMT[i], where='post')
    ax[i].set_ylabel('state')
    ax[i].set_xlabel('time')
```



**g)** Utilizando la secuencia de estados de la cadena de Markov en el régimen estacionario:

- Estima la distribución estacionaria del proceso de saltos subyacente a partir de una única trayectoria de la cadena de Markov en tiempo continuo.
- Estima la distribución estacionaria de la cadena de Markov en tiempo continuo a partir de una única trayectoria del proceso.

- Estima la distribución de la cadena de Markov en tiempo continuo en el límite  $t \rightarrow \infty$  a partir de los estados finales en  $M = 1000$  trayectorias simuladas.
- Comenta los resultados de los apartados anteriores.

```
In [3]: # Stationary distribution of the jump process from a single trajectory
times, trajectories = arrival_times_CTMT[0], trajectories_CTMT[0]
# Count states.
_, c = np.unique(trajectories, return_counts = True)
c = c/np.sum(c)
print("Distribución estacionaria del proceso de saltos:", c)
```

Distribución estacionaria del proceso de saltos: [0.2 0.4 0.4]

```
In [4]: # Stationary distribution of the CTMC from a single trajectory
# Transform arrays to numpy arrays.
trajectories = np.array(trajectories)
times = np.array(times)
# Append end time to arrival times.
times = np.append(times, t1)

# Compute those positions (jumps) where the simulation is at each state.
# E.j. Jumps[0] contains those jumps where the simulation got to 0.
jumps = [np.where(trajectories == state)[0]
          for state in range(len(lambda_rates))]

# The total time in each state can be computed as the sumation of
# the times when the state is left minus those where the state is
# arrived.
time = [np.sum(times[a+1]) - np.sum(times[a]) for a in jumps]
print("Distribución estacionaria de la cadena de Markov:", time/sum(time))
```

Distribución estacionaria de la cadena de Markov: [0.14955017 0.68516559 0.16528424]

```
In [5]: # Stationary distribution of the CTMC from the final state in a sample
# of M trajectories
arrival_times_CTMT, trajectories_CTMT = (
    pe.simulate_continuous_time_Markov_Chain(
        transition_matrix, lambda_rates,
        state_0, 10000, t0, t1))
```

```
In [6]: final_states = [trajectories_CTMT[a][-1] for a in
                        range(len(trajectories_CTMT))]
_, c = np.unique(final_states, return_counts = True)
c = c/np.sum(c)
print("Distribución límite de la Cadena de Markov:", c)
```

Distribución límite de la Cadena de Markov: [0.1575 0.6282 0.2143]

Inspeccionando los resultados, vemos que la distribución estacionaria del proceso de saltos obtenida se corresponde con la teórica calculada anteriormente.

Por otro lado, las dos últimas distribuciones empíricas calculadas son estimaciones de la misma distribución, la distribución límite de la cadena en tiempo continuo, que sabemos que coincide con la distribución estacionaria de la misma.

Vemos entonces que los resultados empíricos son similares, parecidos a su vez a los resultados teóricos que habíamos obtenido.

**h)** ¿Coinciden las distribuciones estacionarias de una cadena de Markov en tiempo discreto con el mismo diagrama de transición y la derivada para el proceso en tiempo continuo? En caso de que coincidan indica la razón. En caso de que no coincidan, define una cadena de Markov en tiempo

continuo con el mismo diagrama de transiciones que tenga la misma distribución estacionaria que la correspondiente en tiempo discreto.

Las distribuciones estacionarias no coinciden ya que la distribución estacionaria en tiempo discreto  $(\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_3)$  coincide con aquella en tiempo continuo  $(\pi_1, \pi_2, \pi_3)$  salvo por una relación de proporcionalidad:

$$\pi_i \propto \frac{\tilde{\pi}_i}{\lambda_i}, i = 1, 2, 3.$$

Debido a las restricciones de normalidad en las distribuciones, ambas serán iguales cuando  $\lambda_i = \lambda_j, \forall i, j$  (condición suficiente).

Vemos que cuando los parámetros  $\lambda$  son iguales:

$$\pi_i = \frac{\frac{\tilde{\pi}_i}{\lambda}}{\sum_{i=1}^3 \frac{\tilde{\pi}_i}{\lambda}} = \frac{\tilde{\pi}}{\sum_{i=1}^3 \tilde{\pi}_i} = \tilde{\pi}_i, i = 1, 2, 3.$$

Podemos probar esto empíricamente:

```
In [7]: lambda_rates = [2, 2, 2]

arrival_times_CTMT, trajectories_CTMT = (
    pe.simulate_continuous_time_Markov_Chain(
        transition_matrix, lambda_rates,
        state_0, 1000, t0, t1))

final_states = [trajectories_CTMT[a][-1] for a in
                 range(len(trajectories_CTMT))]
_, c = np.unique(final_states, return_counts = True)
c = c/np.sum(c)
print("Distribución límite de la Cadena de Markov en tiempo continuo:", c)
```

Distribución límite de la Cadena de Markov en tiempo continuo: [0.207 0.396 0.397]

Donde obtenemos unos resultados razonablemente similares a los obtenidos en la distribución estacionaria de la cadena en tiempo discreto.

## Exercise 2: Simulation of a Brownian bridge process

### Expected value of the standard Wiener process

Consider the expression for the standard Brownian bridge

$$BB_{std}(t) = W(t) - W(1)t$$

The mean of this process is

$$\mathbb{E}[BB_{std}(t)] = \mathbb{E}[W(t) - W(1)t]$$

The expected value operator is linear. Therefore,

$$\mathbb{E}[BB_{std}(t)] = \mathbb{E}[W(t)] - \mathbb{E}[W(1)]t.$$

Since process  $W(t)$  has zero mean  $\mathbb{E}[W(t)] = 0, \forall t \geq 0$

$$\mathbb{E}[BB_{std}(t)] = 0, \quad \forall t \in [0, 1].$$

## Covariance function for the standard Brownian bridge

The covariance function for the standard Brownian bridge is defined as

$$\gamma(t, s) = \mathbb{E}[(BB_{std}(s) - \mathbb{E}[BB_{std}(s)])(BB_{std}(t) - \mathbb{E}[BB_{std}(t)])].$$

Using

$$BB_{std}(t) = W(t) - W(1)t, \quad \mathbb{E}[BB_{std}(t)] = 0,$$

and  $\gamma(s, t) = \mathbb{E}[(W(s) - W(1)s)(W(t) - W(1)t)] = \mathbb{E}[W(s)W(t)] - \mathbb{E}[W(s)W(1)t] - \mathbb{E}[W(1)W(t)s] + \mathbb{E}[W(1)^2st]$

- $\mathbb{E}[W(s)W(1)t] = t\mathbb{E}[W(s)] = 0$ .

Finally, taking into account that

$$\mathbb{E}[W(s)W(t)] = \min(s, t),$$

one gets

$$\gamma(s, t) = \min(s, t) + st - ts - st = \min(s, t) - st.$$

## Mean function for a general Brownian bridge

Consider the expression for the Brownian bridge

$$BB(t) = B(t) + (BB_1 - B(t_1)) \frac{t - t_0}{t_1 - t_0}.$$

The mean of this process is

$$\mathbb{E}[BB(t)] = \mathbb{E}[B(t)] + (BB_1 - \mathbb{E}[B(t_1)]) \frac{t - t_0}{t_1 - t_0}.$$

Using the fact that

$$B(t) \sim \mathcal{N}(BB_0 + \mu(t - t_0), \sigma\sqrt{t - t_0}),$$

and  $\mu = 0$ , the expected value is  $\mathbb{E}[B(t)] = BB_0 \forall t \in [t_0, t_1]$ . Hence,

$$\mathbb{E}[BB(t)] = BB_0 + (BB_1 - BB_0) \frac{t - t_0}{t_1 - t_0}.$$

## Covariance function for a general Brownian bridge

Using the brownian formula

$$BB(t) = B(t) + (BB_1 - B(t_1)) \frac{t - t_0}{t_1 - t_0},$$

and the fact that the covariance function satisfies

$$\text{Cov}(aX + b, cY + d) = ac\text{Cov}(X, Y).$$

The covariance for the Brownian bridge reduces to

$$\begin{aligned}\gamma(t, s) &= \text{Cov}\left(B(t) + \frac{t - t_0}{t_1 - t_0} B(t_1), B(s) + \frac{s - t_0}{t_1 - t_0} B(t_1)\right) \\ &= \text{Cov}(B(t), B(s)) - \frac{s - t_0}{t_1 - t_0} \text{Cov}(B(t), B(t_1)) - \frac{t - t_0}{t_1 - t_0} \text{Cov}(B(s), B(t_1)) + \frac{t - t_0}{t_1 - t_0} \frac{s - t_0}{t_1 - t_0} \text{Cov}(B(t_1), B(t_1))\end{aligned}$$

Using the covariance of the Brownian, that is,  $\text{Cov}(B(t), B(s)) = \sigma^2 \min\{t - t_0, s - t_0\}$ , the covariance for the Brownian bridge results

$$\begin{aligned}\gamma(t, s) &= \sigma^2 \min\{t - t_0, s - t_0\} - \sigma^2 \frac{(s - t_0)(t - t_0)}{t_1 - t_0} - \sigma^2 \frac{(s - t_0)(t - t_0)}{t_1 - t_0} + \sigma^2 \frac{(s - t_0)(t - t_0)}{(t_1 - t_0)^2} \\ &= \sigma^2 \min\{t - t_0, s - t_0\} - \sigma^2 \frac{(s - t_0)(t - t_0)}{t_1 - t_0}\end{aligned}$$

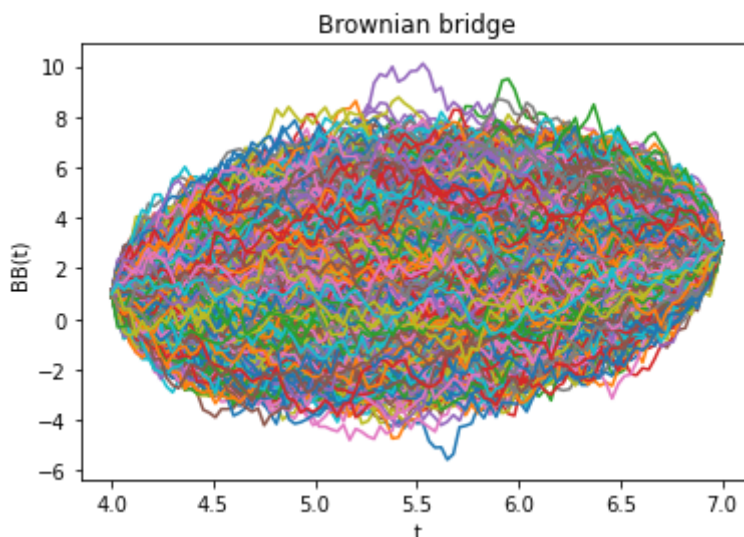


## Brownian bridge covariance simulation

We estimate the covariance value  $\gamma(t, t_{ref})$  of a Brownian bridge verifying  $BB(t_0 = 4.0) = 1.0, BB(t_1 = 7.0) = 3.0, \sigma = 2.0$  with  $t_{ref} = 5.3$  using  $M = 10000$  independent simulations of  $N = 100$  iterations.

```
In [8]: ## Brownian bridge simulation

t0, B0, t1, B1, sigma = 4.0, 1.0, 7.0, 3.0, 2.0
M, N = 10000, 100
t, BB = BM.simulate_Brownian_bridge(t0, B0, t1, B1, sigma, M, N)
plt.plot(t, BB.T)
plt.xlabel("t")
plt.ylabel("BB(t)")
_ = plt.title("Brownian bridge")
```



```
In [9]: # Compute nearest upper position to desired point 5.3
position = np.where(t >= 5.3)[0][0]

# Compute the bias of the Brownian, i.e., BB(t) - E[BB(t)] for each
# t and each simulation.
global_bias = BB - np.array([np.mean(BB, axis = 0),]*BB.shape[0])

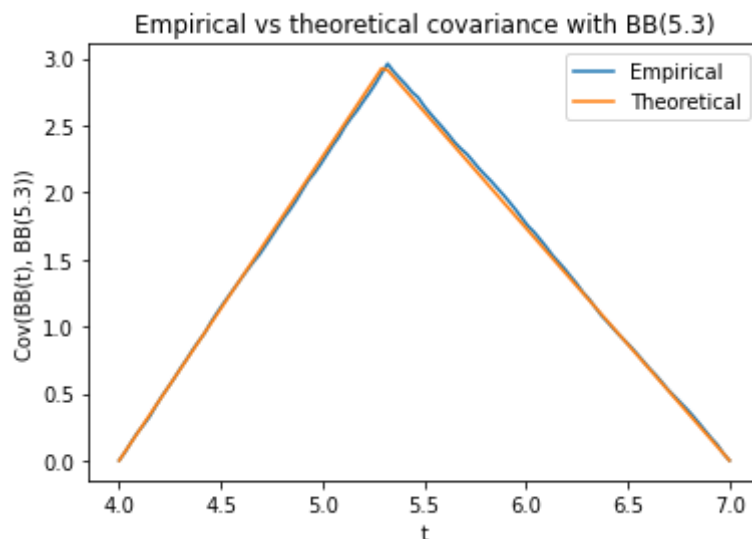
# Compute the bias of the Brownian bias at the desired position.
position_bias = global_bias[:,position]

# Define an auxiliar function to compute the theoretical covariance.
```

```
def gamma(r,s, t0, t1, sigma=2):
    return sigma**2 * np.min([r-t0, s-t0]) - sigma**2*(r-t0)*(s-t0)/(t1-t0)

# Compute the theoretical covariance
theoretical = [gamma(a, 5.3, t0, t1) for a in t]

# Simultaneously plot theoretical and empirical covariance.
plt.plot(t, global_bias.T@position_bias/M, label = "Empirical")
plt.plot(t, theoretical, label = "Theoretical")
plt.title("Empirical vs theoretical covariance with BB(5.3)")
plt.xlabel("t")
plt.ylabel("Cov(BB(t), BB(5.3))")
_ = plt.legend()
```



## Exercise 3: Simulation of a stochastic volatility model

### Derivation of $\mathbb{E}[\sigma(t)]$

Consider the Ito SDE

$$d\sigma(t) = -\alpha(\sigma(t) - \sigma_\infty)dt + \xi dW(t),$$

where  $\alpha > 0$  is the reversion rate,  $\xi > 0$  is the volatility of the stochastic process  $\sigma(t)$ .

Define the process  $X(t) = \sigma(t) - \sigma_\infty$ . Since  $\sigma_\infty$  is constant,  $dX(t) = d\sigma(t)$ , and the equation becomes

$$dX(t) = -\alpha X(t)dt + \xi dW(t).$$

Let's take the expected value on both sides of the equation

$$\mathbb{E}[dX(t)] = \mathbb{E}[-\alpha X(t)dt + \xi dW(t)].$$

Using the fact that the expected value is a linear operator, we get

$$d\mathbb{E}[X(t)] = -\alpha\mathbb{E}[X(t)]dt + \xi\mathbb{E}[dW(t)].$$

Since the Wiener process has zero mean, the equation becomes

$$d\mathbb{E}[X(t)] = -\alpha\mathbb{E}[X(t)]dt.$$



Using the method of separation of variables

$$\frac{1}{\mathbb{E}[X(t)]} d\mathbb{E}[X(t)] = -\alpha dt,$$

the equation can be integrated between  $(t_0, X_0)$ , where  $X_0 = \sigma_0 - \sigma_\infty$ , and  $(t, \mathbb{E}[X(t)])$

$$\int_{X_0}^{\mathbb{E}[X(t)]} \frac{1}{\mathbb{E}[X]} d\mathbb{E}[X] = -\alpha \int_{t_0}^t d\tau.$$

The solution is

$$\log \frac{\mathbb{E}[X(t)]}{X_0} = -\alpha(t - t_0).$$

After some straightforward algebraic manipulation, one obtains

$$\mathbb{E}[X(t)] = X_0 e^{-\alpha(t-t_0)}.$$

Undoing the change of variable, we obtain

$$\mathbb{E}[\sigma(t)] = \sigma_\infty + (\sigma_0 - \sigma_\infty) e^{-\alpha(t-t_0)}.$$

## Derivation of $\text{Var}[\sigma(t)]$

**d)** We apply Itô's lemma to X's SDE

$$dX(t) = a(t, X(t))dt + b(t, X(t))dW(t) = -\alpha X(t)dt + \xi dW(t)$$

obtaining:

$$Y(t) = \phi(t, X(t)) = X(t)^2 \implies dY(t) = \tilde{a}(t, X(t))dt + \tilde{b}(t, X(t))dW(t).$$

Where

$$\begin{cases} \tilde{a}(t, X(t)) &= \frac{\partial}{\partial t} \phi(t, X(t)) + \frac{\partial}{\partial x} \phi(t, X(t)) a(t, X(t)) + \frac{1}{2} \frac{\partial^2}{\partial x^2} \phi(t, X(t)) b(t, X(t))^2 \\ \tilde{b}(t, X(t)) &= \frac{\partial}{\partial x} \phi(t, X(t)) b(t, X(t)) \\ \frac{\partial}{\partial t} \phi(t, X(t)) &= 0 \\ \frac{\partial}{\partial x} \phi(t, X(t)) &= 2X(t) \\ \frac{\partial^2}{\partial x^2} \phi(t, X(t)) &= 2 \end{cases}$$

Summarizing the above formulas, Y's SDE takes the form:

$$dY(t) = (-2\alpha Y(t) + \xi^2)dt + 2\xi \sqrt{Y(t)} dW(t)$$

$$Y(0) = X(0)^2 = (\sigma_0 - \sigma_\infty)^2$$

**e)** Applying expectation to the above formula and using  $\mathbb{E}[dW(t)] = 0$ ,

$$\begin{aligned} d\mathbb{E}[Y(t)] &= \mathbb{E}[dY(t)] = (-2\alpha \mathbb{E}[Y(t)] + \xi^2)dt \implies \frac{d\mathbb{E}[Y(t)]}{-2\alpha \mathbb{E}[Y(t)] + \xi^2} = dt \implies \int_{Y_0}^{\mathbb{E}[Y(t)]} \frac{1}{-2\alpha} \\ &= \frac{\xi^2}{2\alpha} + \left( Y_0 - \frac{\xi^2}{2\alpha} \right) e^{-2\alpha(t-t_0)} = \frac{\xi^2}{2\alpha} + \left( (\sigma_0 - \sigma_\infty)^2 - \frac{\xi^2}{2\alpha} \right) e^{-2\alpha(t-t_0)} \end{aligned}$$

**f)** Using both equations:

$$\text{var}(\sigma(t)) = \mathbb{E}[\sigma(t)^2] - \mathbb{E}[\sigma(t)]^2.$$

From section **c)** we know that

$$\mathbb{E}[\sigma(t)]^2 = (\sigma_\infty + (\sigma_0 - \sigma_\infty) e^{-\alpha(t-t_0)})^2.$$

On the other hand, using section **e)**, the other term can be computed using both  $\mathbb{E}[Y(t)]$  and  $\mathbb{E}[\sigma(t)]$ .

$$\mathbb{E}[\sigma(t)^2] = \mathbb{E}[(\sigma(t) - \sigma_\infty)^2 + 2\sigma_\infty\sigma(t) - \sigma_\infty^2] = \mathbb{E}[Y(t)] + 2\sigma_\infty\mathbb{E}[\sigma(t)] - \sigma_\infty^2$$


Summarizing all the information:

$$\begin{aligned} \text{Var}(\sigma(t)) = \frac{\xi^2}{2\alpha} + \left( (\sigma_0 - \sigma_\infty)^2 - \frac{\xi^2}{2\alpha} \right) e^{-2\alpha(t-t_0)} + 2\sigma_\infty \left( \sigma_\infty + (\sigma_0 - \sigma_\infty) e^{-\alpha(t-t_0)} \right) - \sigma_\infty^2 \\ - \frac{\xi^2}{2\alpha} e^{-2\alpha(t-t_0)} \end{aligned}$$

## Derivation of the conditional pdf: $\text{pdf}[t, \sigma(t) | t_0, \sigma_0]$

**g)** Using that the evolution of the process is defined by a Wiener evolution which is Gaussian, the process itself results Gaussian. Its probability density is totally defined by its mean and variance as:

$$\text{pdf}(\sigma(t) = \sigma | \sigma(t_0) = \sigma_0) \equiv \mathcal{N} \left( \sigma | \sigma_\infty + (\sigma_0 - \sigma_\infty) e^{-\alpha(t-t_0)}, \frac{\xi^2}{2\alpha} - \frac{\xi^2}{2\alpha} e^{-2\alpha(t-t_0)} \right)$$

Where the explicit form is omitted (substitute in  $\mathcal{N}(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2} \frac{(x-\mu)^2}{\sigma^2}}$ ). 

## Simulation of the SDE \$

$d\sigma(t) = -\alpha(\sigma(t) - \sigma_\infty) dt + \xi dW(t),$   
\$

**h)** Utilizamos la función "euler\_maruyana" realizada en las prácticas para pintar las trayectorias. Para definir el intervalo de confianza al 99% utilizamos que la distribución es normal, que contiene al 99% de su densidad en  $(\mu - 3\sigma, \mu + 3\sigma)$ .

```
In [10]: # Simulation of a sample of trajectories
t0, T = 2.5, 17.5
sigma0, sigmainf = 2.0, 0.5
alpha = 0.3
xi = 0.2

def a(t, x): return -1*alpha*(x - sigmainf)
def b(t, x): return xi
```

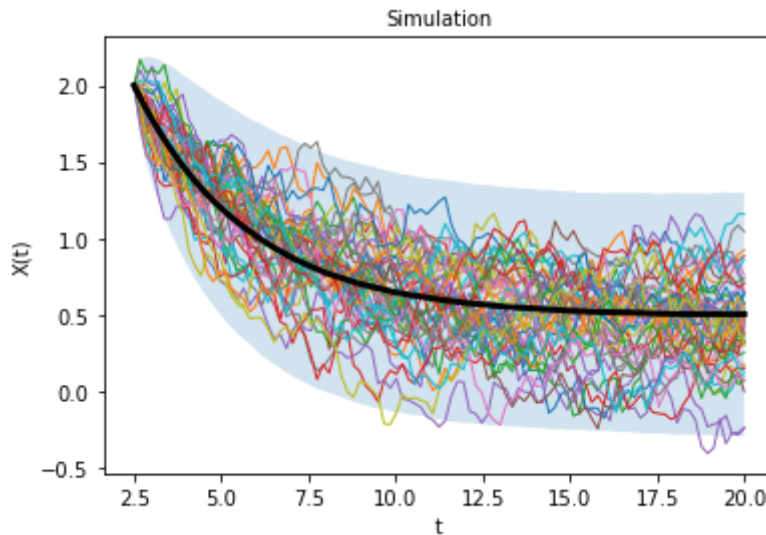
```

t, X = pe.euler_maruyana(t0, sigma0, T, a = a,
                        b = b, M = 10000, N = 100)

stoch.plot_trajectories(t, X, max_trajectories = 50)
# Intervalo
plt.fill_between(t, np.mean(X, axis = 0) - 3*np.std(X, axis = 0),
                np.mean(X, axis = 0) + 3*np.std(X, axis = 0), alpha = 0.2)

```

Out[10]: <matplotlib.collections.PolyCollection at 0x1a0b7ad4c10>

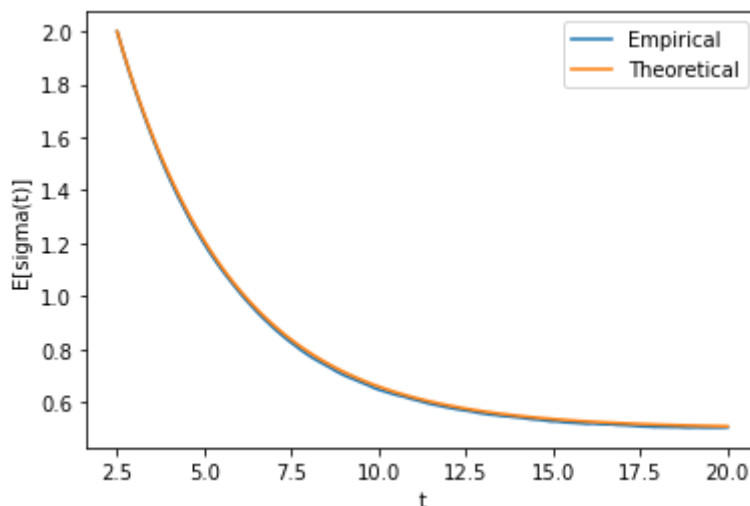


Mostramos la estimación de la media junto con la media teorica.

```

In [11]: plt.plot(t, np.mean(X, axis = 0), label = "Empirical")
def t_mean(t):
    return sigmainf + (sigma0 - sigmainf)*np.exp(-1*alpha*(t-t0))
_ = plt.plot(t, [t_mean(a) for a in t], label = "Theoretical")
plt.xlabel("t")
plt.ylabel("E[sigma(t)]")
_ = plt.legend()

```



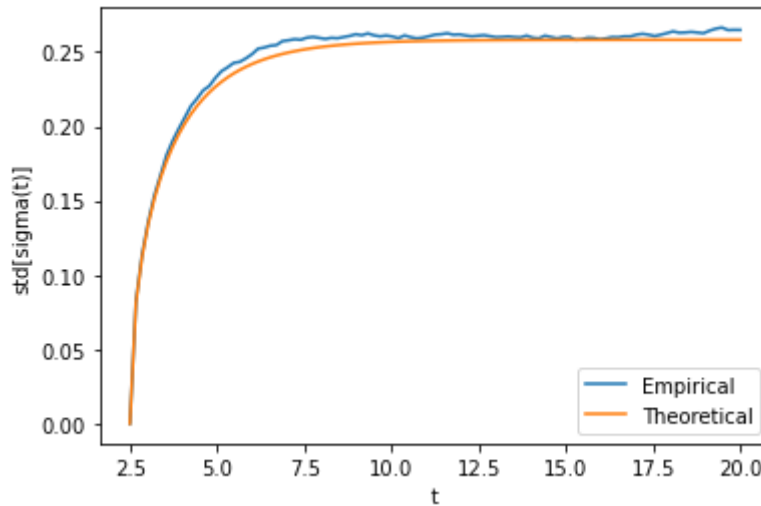
Mostramos la evolución de la desviación típica.

```

In [12]: plt.plot(t, np.std(X, axis = 0), label = "Empirical")
def t_std(t):
    return np.sqrt((xi**2 / (2*alpha)) * (1-np.exp(-2*alpha*(t-t0))))
_ = plt.plot(t, [t_std(a) for a in t], label = "Theoretical")
plt.xlabel("t")

```

```
plt.ylabel("std[sigma(t)]")
_ = plt.legend()
```

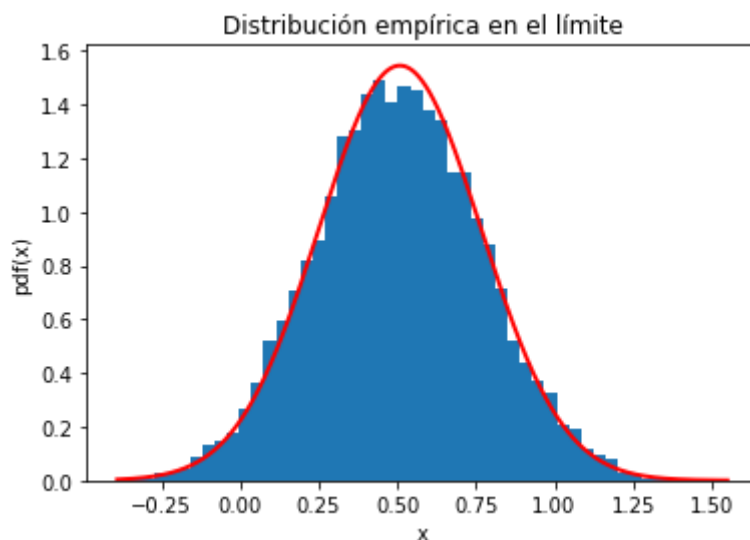


Gráfica de la pdf empírica (histograma normalizado) de  $\sigma(t)$  en el límite y comparación con la pdf teórica.

```
In [13]: x = X[:, -1]
def pdf(x): return sp.stats.norm.pdf(x, loc = t_mean(20), scale = t_std(20))

stoch.plot_pdf(x, pdf)
plt.title("Distribución empírica en el límite")
```

Out[13]: Text(0.5, 1.0, 'Distribución empírica en el límite')



## Exercise 4: Pricing of a European call option

### Pricing by quadrature

```
In [14]: # Parameters that characterize underlying
S0 = 100.0
sigma = 0.3

# Parameters of the EU call option
K = 90.0
T = 2.0
```

```
# Risk-free interest rate  
r = 0.05  
  
price_EU_call = pe.price_EU_call(S0, K, r, sigma, T)  
  
print('Price = {:.4f}'.format(price_EU_call))
```

Price = 26.2402

## Monte Carlo pricing of a European call option

In [ ]:

In [ ]:

In [ ]: